RSRE
MEMORANDUM No. 4261

ROYAL SIGNALS & RADAR ESTABLISHMENT

CAUSING AND PREVENTING VIRUSES IN COMPUTER SYSTEMS

Author: S R Wiseman

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

CONDITIONS OF RELEASE

0037443 BR-109824

..... U

..... Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4261

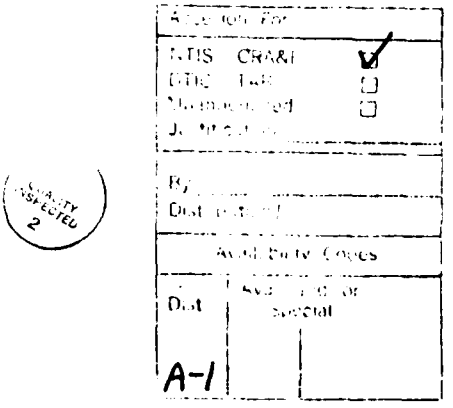
Title: Causing and Preventing Viruses in Computer Systems

Author: S R Wiseman

Date: January 1989

ABSTRACT

Viruses may attack computer systems and carry with them a variety of symptoms. Details of the many ways in which they spread are given and it is shown how this is prevented in conventional systems using procedural controls. More effective measures, which are to be employed in the SMITE secure system, are also described.



1. Introduction

This paper is about software viruses. These reside in programs on disc and, when the program is run, propagate themselves and spread to other programs. Apart from duplication, a virus can cause a wide variety of symptoms to appear, ranging from the benign to the catastrophic. If computers form part of a distributed system, a virus may propagate itself across the network to infect software on another machine, forming a plague.

Since a virus is just a piece of code which hides itself inside the code of a program, it is very difficult to identify infected programs. In any event programs used to examine and heal other programs may themselves be infected, so the virus can spread as fast as it is eliminated.

This paper details how a virus can propagate through a system, so that the enormity of the problem can be appreciated. Next, some measures are suggested which largely overcome the problem in conventional system architectures. The SMITE system, which offers a more complete solution, is then described. This is a system which has been designed for high assurance multi-level secure applications [Wiseman86].

2. Viral Infections

An infection is introduced into a system by persuading an authorised user of the system to run an infected program. This is usually done by an application writer or distributor planting the virus in copies of new programs. When the legitimate users execute these programs the virus spreads. The initial viral spore may disguise the source of the infection by erasing itself from the original program.

A virus propagates by altering programs, either directly by changing their code on disc or indirectly by altering the association between names and programs, which is usually held in a directory. This latter method is usually overlooked, and even excluded by some definitions [Cohen84], but in most systems it is easier to exploit.

The programs in question may be executable images, bootstrap loaders, the operating system image, command files or any other list of instructions. A virus can only propagate if, when it executes, it has the ability to alter these programs or the directories that name them. It follows that a virus placed in a bootstrap or operating system image is unstoppable, because it has the ability to alter any part of the disc. On systems that offer no file protection, a virus in an ordinary program can also propagate easily.

Using the direct approach to propagation (figl), a virus alters the instructions that constitute the uninfected program. The program is modified so that the code of the virus is executed before the program's usual function is carried out. In this way the program appears to behave normally.

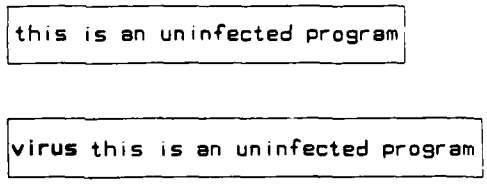


Fig 1: A virus can propagate by altering programs.

With the indirect approach (fig2), a virus alters the mapping between the identifiers that users use to name programs and the programs' code. Thus rather than alter a program in situ, a new program is created by copying the original along with the virus code. This new program is then given the same name as the original, so in future users will run the infected program.

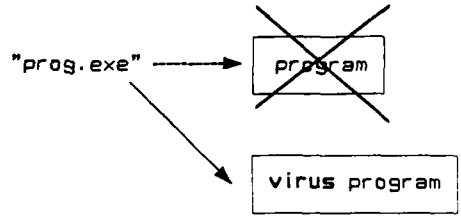


Fig2: A virus can propagate by altering names.

On some systems it is possible to chain from one program to another. In this case the virus need not copy the original program. Instead, the original program is given another name and a new program, which contains the virus code and chains to the original program, is created (fig3). This new program is then given the name of the original. This technique really only works if the original program, with the new name, goes undetected. However this is usually simple on big systems where there are many system files that have strange names and one more will not be noticed.

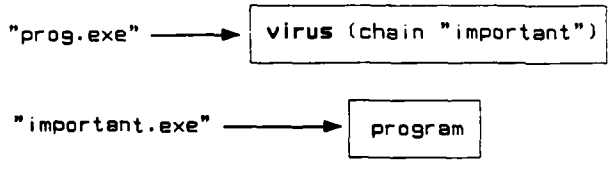


Fig3: A virus can propagate by changing names and chaining.

A virus is more likely to go unnoticed, especially on small systems, if its code is small. A sudden increase in disc space used is an indication that a virus is at work. However viruses can be very small and could occupy the spare bytes in the file which inevitably arise when the program's size is rounded up to a whole number of disc sectors.

A virus must make sure that it does not infect programs that are already infected. If this happens disc space will be eaten away and the programs will take longer to execute as each infection must run before the main program. However, a virus will generally only be able to determine whether the latest infection given to a program is a copy of itself. A virus cannot detect whether a program is infected by another virus. Thus if two or more viruses attack a system at the same time, program sizes will grow as each virus in turn infects them.

When an infected program is executed, the virus may propagate itself, in the manner just described, and produce various side effects. Essentially the virus inherits all the rights and privileges of the user who executes it, though a virus in a bootstrap can of course do anything. In this position it can delete or alter files, release sensitive information to others or masquerade as the user and request services or send fake messages. The possibilities are endless. Usually the virus will not activate itself in an obvious manner until some time has passed, perhaps after a certain number of generations, in order that the virus may be spread widely before being detected.

3. Conventional Safeguards

This section considers how protection facilities offered by conventional systems can be used to protect against viruses.

The best place for a virus is in the bootstrap or system image on disc. If the system has no file protection, these cannot be protected from a viral spore in an ordinary program. In fact, such systems are wide open to attack and nothing can be done apart from preventing the entry of the viral spore by procedural means. That is always use write protect tabs whenever possible and never run programs which have not come from a reputable dealer and hope that they have not been infected inadvertently.

In some systems the bootstrap will pull in a system image from a floppy disc or tape if one is present in the drive when the system is booted. In this case the program on the floppy disc is executed with full access to the machine including the main disc. Therefore, regardless of the protection offered by the operating system on the main disc, a virus or viral spore on the floppy disc can propagate to the main disc. The only safeguard here is again procedural: ensure that the system is never booted while a floppy disc is present in the drive.

Note that, if the system does not extend file protection to floppy discs, an ordinary program could write a bootstrap to any floppy disc in the drive. Therefore the user should not consider some floppy discs to be safe because they are believed to not contain a bootstrap.

Computers in distributed systems may load themselves from a boot server. To do this they broadcast a request on the network asking for a system image. The boot server responds to such requests by sending a copy of the operating system to the requestor. However there is nothing to stop other computers masquerading as boot servers. They may send infected system images to the requestor before the proper boot server responds.

A possible solution to this is to place the boot server and systems that use it behind an intelligent network bridge which filters out data sent to machines which are not fully booted. Once booted these machines change their network address to one that the bridge lets through. This protects the machines behind the bridge from attack by general machines on the network until they are able to fend for themselves.

In systems that offer file protection more can be done to combat the spread of a virus. Firstly, an infected program executed by a user is prevented from propagating the virus to programs which the user cannot modify or rename. Protection of this sort is usually found in multi-user systems.

Simple protection facilities, like read only and execute only access rights, do not in fact prevent a program being contaminated by a virus inadvertently executed by the program's owner. This is because the virus, which inherits the owner's privileges, simply alters the program's access rights to allow modification. Once the virus has been inserted the access rights can be restored as if nothing has happened.

Some more sophisticated operating systems have protection mechanisms which can be used to prevent the owner of a program from modifying its protection attributes. In these systems, programs should be set so that no user can modify them or alter their protection. This prevents a virus from spreading using the direct method, but not the indirect method.

The indirect method of spreading a virus relies on the ability to alter how programs are named. Operating systems which offer file protection include limited protection of directories. Using these facilities it is possible to prevent a virus spreading to programs named in directories which cannot be accessed by the user. However, users are generally allowed to change their own directories, because this is how they create new files and delete old ones. Therefore a virus can spread, using the indirect method, to any program named in the user's directory.

If the operating system's file protection mechanism could be used to prevent a program from being renamed, this would offer a partial solution to the problem of indirect propagation. However this would make program maintenance impossible.

The problem arises because directories are fundamental to the operation of conventional systems, and are readily exploited by viruses. Therefore, little can be done to protect the user who executes software of dubious origins.

Procedural safeguards can be imposed to ensure that viruses do not spread to the programs which are supplied as common system utilities in multi-user systems. Simple file protection mechanisms prevent ordinary users spreading a virus to these, but special safeguards are needed to prevent their owner, usually the system manager, from doing so.

The system manager must not execute any program which could contain a virus or viral spore. The easiest way to ensure this is to limit the system manager to executing just two programs, one to remove old programs and one to install new ones. These programs would be supplied with the system and are, hopefully, virus free. The system manager should not install any program which is of dubious origin, because this could infect the programs of the users who run it. Under no account should the system manager execute any other program, especially those of other users, as this may spread a virus.

Other mechanisms,based upon encryption, have been proposed for combatting viruses, for example [Pozzo&Gray87]. These aim to detect changes to programs either by encrypting them or attaching cryptographic checksums to them. However these methods have a serious effect on performance and do not protect from indirect virus attacks.

4. SMITE's Defence System

The SMITE secure system, and the Flex environment on which it is based [Foster*et.al*82], does not use directories to access files. Instead unforgeable, unalterable addresses, or capabilities [Wiseman88a], are used to directly address objects such as text and programs. Editors, compilers and other programs are able to manipulate these capabilities and store them on disc along with other data.

A capability based backing store can be used to support a modular compilation system [Harrold88] and editors for structured text [Core87], without the need for names and directories [Stanley85a]. However, directories are required for naming some text files which contain capabilities for programs. Such a system is still, therefore, vulnerable to virus attacks.

The SMITE backing store [Wiseman88b] is organised on a write once basis. That is, when data is stored in the backing store, some free space is found to accommodate it. A capability which refers to the data is created and returned as a result of the store operation. The data is retrieved by applying a load operation to the capability. Note that without the capability, the data cannot be retrieved and capabilities are themselves protected and cannot be forged.

Programs are free to use the backing store to pass parameters and save temporary values. Capabilities are used to name backing store data directly, rather than indirectly through a directory. This means programs do not need to be able to alter directories in order to carry out their function. It is therefore practical to consider restricting the ability to alter directories to the user at the command level.

This method of working, where names are used relatively infrequently, is quite different to that of the conventional operating system. However it only works if it is well integrated into the system as a whole. This is the case in the Flex system, which serves as a demonstration that the method is not only feasible but to some extent desirable [Stanley85b].

Restricting directory updating to the command level, prevents a virus from propagating indirectly by giving an infected version of a program the name of the original. It also has the advantage of preventing Trojan Horses [Boebert&Kain85] from exploiting the signalling channels inherent in the use of names in shared directories.

The SMITE system provides the Trusted Path mechanism [Wiseman et.al.88] as a means for software to determine whether it has been invoked directly by the user at the command level, or indirectly by some program that the user has run. The Trusted Path is a command interface which is shown to be free of Trojan Horses, that is it obeys commands if and only if the user directs it to do so. This is part of the secure system and is installed and initialised in a way which cannot be subverted.

The software module which is responsible for manipulating directories may be incorporated into many programs, as well being used by the command level software. It allows software to look up names in a directory at any time. However, before updating a directory it uses the Trusted Path mechanism to ensure that it is being invoked directly as part of the command level.

The use of capabilities to name objects directly, also extends across networks [Foster&Currie86], as does the Trusted Path mechanism. Therefore a virus in a SMITE program is completely contained, and in effect cannot qualify as a virus.

5. Conclusions

A virus can spread either directly, by altering program code, or indirectly by altering how programs are named. In operating systems that have no file protection, nothing can be done to prevent a virus spreading, other than to avoid software of dubious origin. File protection mechanisms, such as those found in multi-user systems, can be used to prevent the spread of viruses from one user to another. However, they do not stop a virus spreading to a user's programs if the user executes an infected program.

A novel method of working is proposed for the SMITE secure system, in which objects in the backing store are rarely accessed by a textual name. Such an environment is successfully employed in the Flex system. This is achieved by offering a backing store that uses capabilities to address data directly, so programs no longer need to be able to update directories in order to communicate. It is sometimes necessary to update textual names, but this is made the direct responsibility of the user using a Trusted Path mechanism, and cannot be performed by a program.

Acknowledgements

Thanks to Antony Martin for help in formulating the ideas presented here and for suggesting a solution to the network booting problem.

References

W E Boebert & R Y Kain,
"Secure Computing: The Secure Ada Target Approach"
Scientific Honeyweller Vol 6 Num 2 July 1985 ppl-17

F Cohen,
"Computer Viruses: Theory and Experiments",
7th National Computer Security Conference,
September 1984, pp240-263
also: Computers & Security, Vol 6, Num 1,
February 1987, pp22..35

P W Core,
"User Extensible Graphics Using Abstract Structure"
RSRE Report 87011, August 1987

J M Foster, I F Currie & P W Edwards,
"Flex: A Working Computer with an Architecture Based on Procedure Values",
Procs. Int. Workshop on High Level Language Computer Architecture,
Fort Lauderdale, Florida, Dec 1982.
(Also RSRE Memo 3500, July 1982)

J M Foster & I F Currie,
"Remote Capabilities in Computer Networks",
RSRE Memo 3947, March 1986

C L Harrold,
"The SMITE Modular Compilation System",
RSRE Memo 4145, March 1988

M M Pozzo & T E Gray
"An Approach to Containing Computer Viruses"
Computers & Security, Vol 6, Num 4
August 1987, pp321..331

M Stanley,
"Extending Data Typing Beyond the Bounds of Programming Environments",
RSRE Memo 3878, September 1985 (a)

M Stanley,
"The Use of Values without Names in a Programming Support Environment",
RSRE Memo 3901, November 1985 (b)

S R Wiseman,
"A Secure Capability Computer System",
Procs. IEEE Symp. on Security and Privacy,
Oakland, CA, April 1986

S R Wiseman,
"Protection and Security Mechanisms in the SMITE Capability Computer",
RSRE Memo 4117, January 1988 (a)

S R Wiseman,
"The SMITE Object Oriented Backing Store",
RSRE Memo 4147, March 1988 (b)

S R Wiseman, P F Terry, A W Wood & C L Harrold
"The Trusted Path between SMITE and the User",
Procs. IEEE Symp. on Security and Privacy,
Oakland, CA, April 1988

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..... UNCLASSIFIED .....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	Memo 4261		U/C

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
7784000	ROYAL SIGNALS & RADAR ESTABLISHMENT ST ANDREWS ROAD, GREAT MALVERN, WORCESTERSHIRE WR14 3PS

5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title

CAUSING AND PREVENTING VIRUSES IN COMPUTER SYSTEMS

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
WISEMAN S			1989.01	8

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement

UNLIMITED

Descriptors (or keywords)

continue on separate piece of paper

Abstract

Viruses may attack computer systems and carry with them a variety of symptoms. Details of the many ways in which they spread are given and it is shown how this is prevented in conventional systems using procedural controls. More effective measures, which are to be employed in the SMITE secure system are also described.